

# Mining Users' Significant Driving Routes with Low-power Sensors

Sarfraz Nawaz  
Computer Laboratory  
University of Cambridge  
sarfraz.nawaz@cl.cam.ac.uk

Cecilia Mascolo  
Computer Laboratory  
University of Cambridge  
cecilia.mascolo@cl.cam.ac.uk

## Abstract

While there is significant work on sensing and recognition of significant places for users, little attention has been given to users' *significant routes*. Recognizing these routine journeys, opens doors to the development of novel applications, like personalized travel alerts, and enhancement of user's travel experience. However, the high energy consumption of traditional location sensing technologies, such as GPS or WiFi based localization, is a barrier to passive and ubiquitous route sensing through smartphones.

In this paper, we present a passive route sensing framework that continuously monitors a vehicle user *solely through a phone's gyroscope and accelerometer*. This approach can differentiate and recognize various routes taken by the user by time warping angular speeds experienced by the phone while in transit and is independent of phone orientation and location within the vehicle, small detours and traffic conditions. We compare the route learning and recognition capabilities of this approach with GPS trajectory analysis and show that it achieves similar performance. Moreover, with an embedded co-processor, common to most new generation phones, it achieves energy savings of an order of magnitude over the GPS sensor.

## Categories and Subject Descriptors

H.4 [Information Systems Applications]: Miscellaneous; I.5.4 [Pattern Recognition]: Applications; C.1.3 [Computer Systems Organization]: Processor Architectures—*Heterogeneous (hybrid) systems*

## General Terms

Experimentation, Algorithms, Measurement

## Keywords

Mobile Sensing, Significant Journeys, Route Sensing

## 1 Introduction

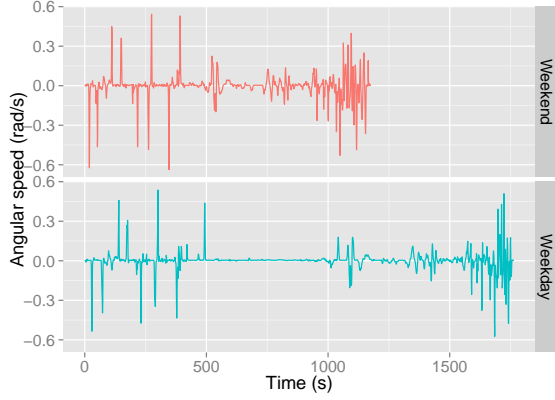
Modern smartphones have a wide range of embedded sensors and are increasingly being used as a novel sensing platform to sense all aspects of a user's life ranging from health and well-being to driving style. They are a perfect tool for sensing user's context and proactively providing information to the user that he or she will find useful like a virtual personal assistant [2]. One form of this context sensing is *place learning* that identifies significant places where a user usually spends some of his or her time regularly, for example, home, work or let's say a gym. While there is significant amount of work on learning these significant places [22, 17, 6], little attention has been given to *significant journeys*, i.e., journeys that users regularly make during their daily routines.

The detection of a user's significant journeys can have various applications in terms of the general quantified self movement: mapping the daily displacements of a person as well as his or her activity has become a fashionable demand in recent years. In addition, the early detection of the fact that a user has started routine journey can improve travel recommendations and traffic updates. We have also seen applications related to preparation of the destination location for user's arrival or departure: smart home heating systems [3] switching on/off when a journey to/from a place is detected, social life alerts to family of the arrival of person, and so on.

However, continuous monitoring of user's journeys can be expensive as location sensing through user's personal device can incur prohibitive energy costs. While it is possible to use duty cycling [31] or trigger location sensing [38] from low power sensors, the energy cost of location sensing is high when location sensors are active. It is also possible for the phone to be plugged in a charging port in the vehicle during journeys but this places an extra constraint on the user to remember to plug the phone in during each journey for energy intensive location sensing. Significantly lowering the energy consumption of sensing these journeys will relieve the user of this requirement and thus make it more likely and convenient for users to adopt and use the system. Therefore, in this paper, we explore the use of alternative low energy phone sensors for accurate detection of user's significant routes. Sensors such as accelerometer and gyroscopes can be reasonably cheap for the detection of user patterns such as change in user activity and means of travel. A large body of literature has covered user activity detec-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SenSys'14, November 3–6, 2014, Memphis, TN, USA.  
Copyright © 2014 ACM 978-1-4503-3143-2 ...\$10.00



**Figure 1. Comparison of processed gyroscope output from a smartphone on two different days for the same route.**

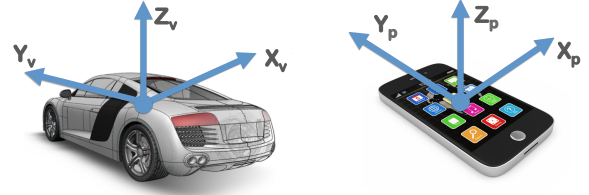
tion [7, 28, 25], transport mode detection [15, 48] and other vehicle and travel related applications [10, 33, 54, 8, 53] using these sensors. The use of these sensors has become even more energy efficient recently, thanks to the advent of embedded co-processors in modern phones such as the iPhones M7 motion co-processor and Qualcomm Hexagon QDSP6 of the Snapdragon 800 processor platform available on Google Nexus 5, Nokia Lumia 1520, Sony Xperia Z1 and Samsung Galaxy Note 3 (LTE).

In this paper, we take advantage solely of accelerometer and gyroscope of a user’s phone to detect with high accuracy significant, i.e. repeated, user journeys. The approach is able to distinguish user routes by employing Dynamic Time Warping of angular speeds experienced by the phone while in transit and is independent of phone’s orientation, location within the vehicle and traffic conditions. In order to show the effectiveness of the approach we compare this route learning framework with one where GPS is used and showing comparable performance if only the CPU is used for sensing and processing and then with an approach which takes advantage of co-processor computation achieving energy savings of an order of magnitude over the GPS sensor.

To our knowledge, this is the first approach of its kind. While inertial navigation can be used to track the position of a vehicle using inertial sensors like accelerometer, gyroscope and magnetometer, the noise characteristics of low cost MEMS inertial sensors [55, 49] in smartphones make them unsuitable for inertial only navigation and tracking. A MEMS sensor based inertial navigation system either requires periodic external information [49, 12] or extensive user training and calibration [13] to be useful.

To summarize, the contributions of this work are the followings

- A Dynamic Time Warping based framework for significant route detection based solely on accelerometer and gyroscope sensors in the phone.
- A comparison of the sensing performance against a GPS based approach. The results show comparable performance.



**Figure 2. Vehicle coordi-** **Figure 3. Phone coordi-**  
**nate system** **nate system**

- A prototype implementation of the system with a low power co-processor.
- A comparison of the energy consumption of co-processor based system with the GPS based approach. The results show one order of magnitude of energy savings over the GPS based approach.

## 2 Approach Overview

In this section we describe how our approach of sensing significant driving routes takes advantage of a phone’s sensors. When a user carrying a smartphone travels in a car, the sensors embedded in the phone, for example, the accelerometer and the gyroscope experience different forces. The accelerometer can sense the acceleration and deceleration of the vehicle and the gyroscope can sense that the vehicle is turning [53]. As the vehicle travels, each turn and bend of the road causes a change in the angular speed sensed by the gyroscope. This creates a unique *signature* of the journey in the form of variations of angular speed. When the user repeats this journey, the phone sensors can recognize the route by observing that the variations in angular speed are similar to the ones of the previous journey. For example, the sequence of turns that a vehicle takes to get from one location to a destination remains the same between different journeys on the same route.

This observation forms the basis of this work. But how repeatable are these patterns and can we really recognize repeated routes from these patterns? Fig. (1) shows two such angular speed patterns captured by the gyroscope of a phone carried in a vehicle from a 12km long route (shown in Fig. 13 as actual route) in a typical urban city when the route is driven on on two different days. The output of the gyroscope has been low-pass filtered and processed to account for phone’s orientation in the vehicle. We will discuss the details in Section 3.1. The upper red trace is from a journey that took place on a weekend and the lower blue trace is from a second journey on the same route in the weekday evening traffic. Two things are evident from this figure: 1) the patterns bear a strong similarity; and 2) these are stretched (or compressed) in time with non-uniform scaling on the time axis. There are some subtle differences too, i.e., some of the angular speed peaks have slightly different magnitudes. The angular speed patterns are similar because the vehicle takes the same route and thus executes same sequence of turns and bends. But since the vehicle travels at different speeds on different parts of the route depending on traffic, this leads to non-uniform scaling of the pattern on the time axis. And as

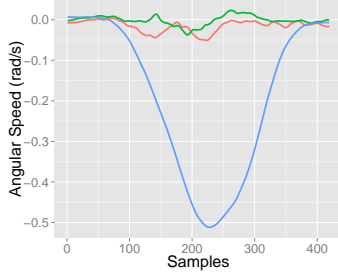


Figure 4. Vehicle aligned

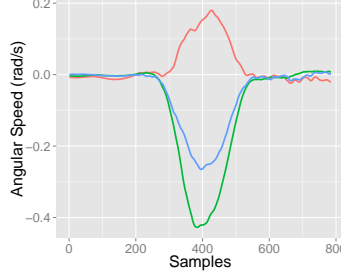


Figure 5. Arbitrary orientation

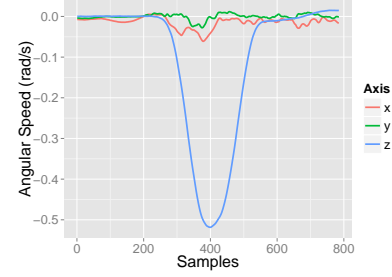


Figure 6. After Z-Alignment

the vehicle can take same turns at different speeds on different occasions, the magnitudes of some of the peaks can be slightly different. Our approach is able to compare the overall similarity of these two patterns under non-uniform time scaling while accounting for the arbitrary way a smartphone can be carried in the vehicle so to recognize repeated routes from the gyroscope sensor in the phone. In the next section we discuss in detail how this can be achieved.

### 3 Approach in Detail

In this section we describe the details of our approach. We first show how we take into account that a user's phone will be oriented arbitrarily. We then describe the dynamic time warping approach which is at the base of the alignment of the journey traces collected with the sensors.

#### 3.1 Z-Axis Alignment

A user usually carries a phone in an arbitrary orientation while traveling in a vehicle. This means that our system cannot use the inertial sensors in the phone to sense turns unless we align the coordinate system used by the phone to a certain orientation. Here we describe how we address this issue. Let us first define two coordinate systems, a vehicle coordinate system and a phone coordinate system. The vehicle coordinate system is defined by three orthogonal axes  $X_v$ ,  $Y_v$  and  $Z_v$  where  $+Y_v$  points in the forward traveling direction,  $+Z_v$  points towards the sky and  $+X_v$  extends to the right as shown in the Fig. (2). Similarly the phone coordinate system is defined by orthogonal axes  $X_p$ ,  $Y_p$  and  $Z_p$  with positive directions as shown in Fig. (3). As a vehicle takes a turn, it rotates on its  $Z_v$  axis but there is almost no rotation around  $X_v$  or  $Y_v$ . This rotation can be sensed by the gyroscope of a phone in the vehicle. Fig. (4) shows the output from the gyroscope as the vehicle carrying the phone takes a right turn. The coordinate system of the phone is aligned to that of the vehicle. It shows that during the turn almost all of the angular speed is contained in the  $z$  component whereas the  $x$  and  $y$  components are negligible because there is little rotation around these axes. This means that we only have to align  $Z_p$  axis of the phone with  $Z_v$  axis of the vehicle to capture its rate of rotation during turns.

Now we describe how we can align  $Z_p$  with  $Z_v$ .  $Z_v$  is in fact aligned with gravity and we can easily estimate the inclination angle  $\theta$  between  $Z_p$  and  $Z_v$  using the accelerometer. Let  $\mathbf{g}_p = [a_x, a_y, a_z]^T$  be a vector estimate of gravity derived from the accelerometer. It is obtained by running a low pass filter on accelerometer output to extract the DC component

in the signal [1, 20]. If  $\hat{\mathbf{g}}_p$  is a unit vector along  $\mathbf{g}_p$  then we can calculate a quaternion  $\mathbf{q}$  [24] that rotates  $Z_p$  to  $Z_v$  through the shortest arc as,

$$\mathbf{q} = \cos \frac{1}{2}\theta + \hat{\mathbf{u}} \sin \frac{1}{2}\theta \quad (1)$$

$$\mathbf{u} = \hat{\mathbf{g}}_p \times \hat{\mathbf{z}} \quad (2)$$

$$\hat{\mathbf{u}} = \frac{1}{\|\mathbf{u}\|} \mathbf{u} \quad (3)$$

$$\mathbf{v} = \hat{\mathbf{g}}_p \cdot \hat{\mathbf{z}} \quad (4)$$

$$\cos \frac{1}{2}\theta = \sqrt{\frac{1+\mathbf{v}}{2}} \quad (5)$$

$$\sin \frac{1}{2}\theta = \sqrt{\frac{1-\mathbf{v}}{2}} \quad (6)$$

where  $\mathbf{u}$  and  $\mathbf{v}$  are cross and dot products respectively and  $\hat{\mathbf{z}} = [0, 0, 1]^T$ . Now we can rotate  $Z_p$  to align it with  $Z_v$  using Hamilton product [24] and calculate the aligned output of the gyroscope  $\omega_r$  as,

$$\omega_r = \mathbf{q} \omega \mathbf{q}^{-1} \quad (7)$$

where  $\omega = [\omega_x, \omega_y, \omega_z]^T$  is the output of the gyroscope and  $\mathbf{q}^{-1}$  is conjugate quaternion,

$$\mathbf{q}^{-1} = \cos \frac{1}{2}\theta - \hat{\mathbf{u}} \sin \frac{1}{2}\theta \quad (8)$$

Here we must point out that we do not align the two coordinate systems completely as in other sensing systems [35, 53]. We only align the  $z$  axis of the phone with the  $z$  axis of the vehicle. The  $x$  and  $y$  axes of the phone,  $X_p$  and  $Y_p$  could be pointing in any direction. However, this does not pose any problems for our application because we only want to sense the angular speed of the vehicle which is contained almost entirely in the  $z$  component. In order to demonstrate this we collect gyroscope data from a phone with an arbitrary orientation in the vehicle as it turns right. Fig. (5) shows the output of the gyroscope. In this case, none of the phone axes are aligned with any of the vehicle axes and this results in components of total angular speed appearing on all three axes of the phone. Fig. (6) shows the gyroscope output after it has been rotated using Eq. (7) of our Z-axis alignment approach. Comparing Fig. (6) with Fig. (4) shows that it is sufficient to align  $Z_p$  with  $Z_v$  to sense vehicle's angular

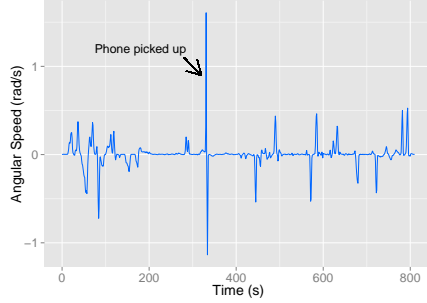


Figure 7. Detecting user interaction with the phone

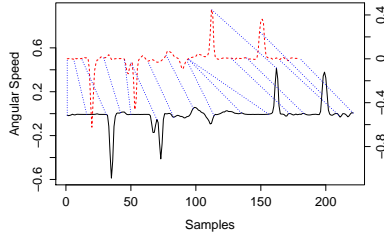


Figure 8. Dynamic time warping

speed and therefore the turns it makes during a journey. Our application and the proposed approach, therefore, does not require complete coordinate alignment. While traveling in a vehicle, users typically carry the phone in their clothing, in a bag or carefully place it in a secure location in the vehicle to avoid any damage. It is, therefore, unlikely for the phone to experience continuous random perturbations during the journey. However, if the user interacts with the phone during the journey, it changes device orientation. But it is easy to detect such an interaction because it causes a large change in gyroscope output. Fig. (7) shows the gyroscope output from a journey where the user picked up the phone while driving and then placed it back. We can use a threshold to detect such an interaction and perform z-axis alignment again.

### 3.2 Dynamic Time Warping

Dynamic time warping is a class of algorithms that calculate dissimilarity or distance between two ordered sequences while allowing compression or expansion of the ordered axis to best align the two sequences. These algorithms map each point in one sequence to one or more points in the second sequence using a dynamic programming approach. Dynamic time warping (DTW) was originally proposed for speech recognition [44, 43] to account for different speaking rates by different speakers. However, since then it has been extensively used for time series analysis, clustering and classification in bioinformatics [16], shape recognition [27], gesture recognition [30] and many other applications.

Let us suppose that we have a time series of angular velocities  $X = \{x_1, x_2, x_3 \dots, x_n\}$  from the gyroscope that has already been aligned with the z-axis of the vehicle. And we want to compare it to a second time series  $Y = \{y_1, y_2, y_3 \dots, y_m\}$  that was captured from a previous journey. If  $i = 1, 2, 3, \dots, n$  and  $j = 1, 2, 3, \dots, m$  are indices into  $X$  and

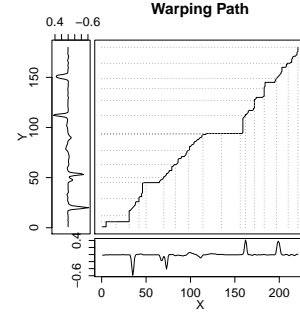


Figure 9. Warping path in warping matrix

$Y$  respectively and  $P$  is an optimal mapping between the two sequences given as,

$$\begin{aligned} P(k) &= (p_x(k), p_y(k)) \\ k &= 1, 2, 3, \dots, T \\ p_x(k) &\in \{1, 2, 3, \dots, n\} \\ p_y(k) &\in \{1, 2, 3, \dots, m\} \end{aligned} \quad (9)$$

we can compute the dissimilarity or distance between the two sequences as,

$$d_{DTW}(X, Y) = \sum_{k=1}^T d(p_x(k), p_y(k))m(k)/M \quad (10)$$

where  $m(k)$  is a per step weight associated with the step pattern discussed later,  $M$  is a normalization constant to account for different lengths of  $X$  and  $Y$  and  $d(i, j) = f(x_i, y_j) \geq 0$  is a non-negative function to compute dissimilarity between individual elements of  $X$  and  $Y$ . The mapping  $P$  is also called a warping path and is calculated by solving the following problem under certain constraints.

$$\min_P d_P(X, Y) \quad (11)$$

These constraints include monotonicity or ordering that can be expressed as,

$$\begin{aligned} p_x(k+1) &\geq p_x(k) \\ p_y(k+1) &\geq p_y(k) \end{aligned} \quad (12)$$

and that the end points of  $X$  are mapped to end points of  $Y$ ,

$$\begin{aligned} p_x(1) &= 1 \\ p_y(1) &= 1 \\ p_x(T) &= n \\ p_y(T) &= m \end{aligned} \quad (13)$$

Dynamic time warping solves the problem given in Eq. (11) using dynamic programming. It constructs a warping matrix  $W$  as,

$$W(i, j) = d(i, j) + S \quad (14)$$

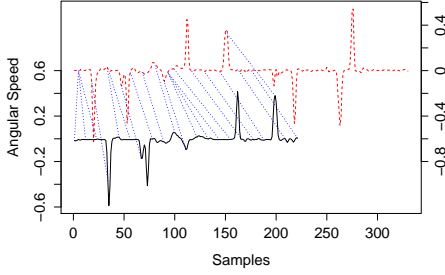


Figure 10. Open ended warping

where  $S$  is called a step pattern and is given as,

$$S = \min \begin{cases} s(i, j-1) \\ s(i-1, j) \\ s(i-1, j-1) \end{cases} \quad (15)$$

$$s(p, q) = \begin{cases} 0 & \text{if } p = 0, q = 0 \\ \infty & \text{if } p = 0, q \neq 0 \\ \infty & \text{if } p \neq 0, q = 0 \\ w(p, q) & \text{otherwise} \end{cases} \quad (16)$$

Starting from  $W(1, 1)$ , this step pattern constructs a warping path that minimizes the dissimilarity between  $X$  and  $Y$  while adhering to the constraints given in Eq. (12) and Eq. (13). When this recursion terminates,  $W(n, m)$  contains the dissimilarity or distance between  $X$  and  $Y$  which can be normalized according to Eq. (10). There are several different types of step patterns [40] that can be used in DTW and additional windowing constraints can also be imposed on  $P$ . However, our experience with our gyroscope dataset indicates that windowing constraints unnecessarily restrict DTW and therefore should not be used for this application. The time complexity of DTW is  $O(nm)$ . Fig. (8) shows a mapping between two sequences computed using DTW. The matched points of two sequences are connected with dotted blue lines. Fig. (9) shows the corresponding wrapping path and the warping matrix where  $i$  grows horizontally towards right and  $j$  grows vertically up.

### 3.3 Open Ended Warping

In the previous section, we presented dynamic time warping based approach to compare two journeys. While it can be used to compare complete journeys, it does not work when comparing a partial journey with a complete previous journey. For example, if we want to compare the output of the gyroscope captured up to the current moment in the journey to a previous journey to predict the destination and remaining route, we cannot use the basic DTW approach because it maps every point between the two sequences  $X$  and  $Y$  and forces end points of  $X$  to map to those of  $Y$ . We, therefore, need a means of comparing a given sequence  $X$  of gyroscope output to all possible truncated versions  $Y^j (j = 1, 2, \dots, m)$  of  $Y$ , gyroscope output from a previous journey, and choose the best possible match i.e.,

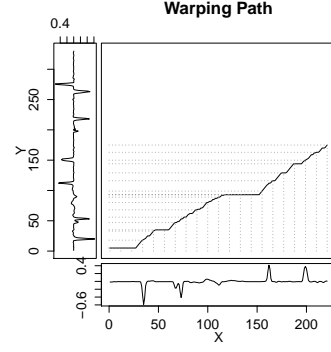


Figure 11. Warping path for open ended matching

$$\min_j d(X, Y^j) \quad (17)$$

We can solve this problem by executing the basic DTW as described in the previous section. And then taking the smallest cumulative distance in  $W(n, j)$  column of the warping matrix instead of  $W(n, m)$ . This gives us the prefix length  $j$ , the distance between  $X$  and  $Y^j$  and the best matching prefix  $Y^j$ . This is known as open-ended DTW [51]. It is also possible to match  $X$  with all possible sub-sequences  $Y^{i,j} (i = 1, 2, \dots, m), j > i$  of  $Y$ ,

$$\min_{i,j} d(X, Y^{i,j}) \quad (18)$$

This is known as open-beginning open-ended DTW [45] and requires only one additional  $n \times m$  matrix to keep track of the starting index of sequences. The time complexity of both open-ended and open-beginning open-ended DTW is  $O(nm)$ . Fig. (10) shows the open-beginning open-ended DTW of two time series of gyroscope data. It shows that it is able to correctly identify the matching portions of the two sequences. Sometimes trips start or end in car parks where turns vary from one trip to another as a user parks in different locations. Open-beginning open-ended DTW is useful in ignoring such turns at the start or end and matching only the intermediate part of the trip. Fig. (11) shows the corresponding warping path and the warping matrix where  $i$  grows horizontally towards right and  $j$  grows vertically up. Fig. (12) shows the DTW mapping between the two complete journeys that we initially discussed in Section 2.

### 3.4 Route Mining

We have described our time warping based approach that is able to distinguish between two routes and is not affected by the type, orientation or position of the sensing device inside the vehicle: we will show results of the evaluation of these aspects in the next sections. We now discuss how we can mine the axis aligned angular speeds captured by the gyroscope to identify important or significant routes that a user frequently travels on.

Let us suppose that during the bootstrap period when a user starts using the system, it captures and aligns the angular speeds of the vehicle from  $n$  journeys made by user. We



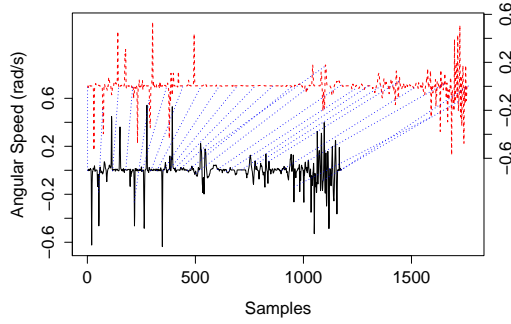


Figure 12. DTW mapping for two example journeys

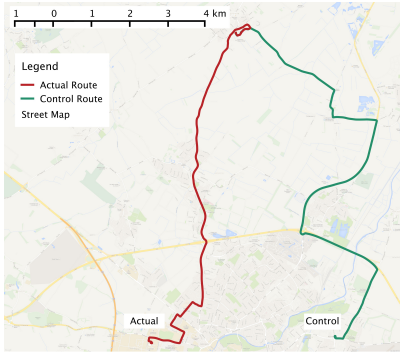


Figure 13. Trajectories of actual and control paths

can then compute normalized DTW distances between each pair of journeys to construct an  $n \times n$  dissimilarity matrix  $D$ . This dissimilarity matrix  $D$  can then be used with any of the several clustering algorithms [18] to group similar journeys or routes together. Clusters with large number of members then represent routes frequently used by the user. In the next few sections we will show how this method performs over a set of collected real traces.

## 4 Accuracy Evaluation

In this section we report the results of the evaluation we conducted to study the feasibility of the approach. We first show how the phone position and type of handset are not affecting the performance. We then show how routes collected through phones handed out to participants can be clustered automatically into significant routes.

### 4.1 Phone Position and Handset Independence

In order to demonstrate that our route matching approach is not affected by phone position inside the traveling vehicle, we conduct a small experiment. We collect a trace of angular speed from a phone inside the vehicle on a typical commute route from source A to destination B. The route is about 12 km long and takes about 20 minutes to complete in ordinary driving conditions. We will refer to this as *actual* route. We also collected a trace from a second route of similar distance from source A to another destination C. We will refer to this as *control* route. Fig. (13) shows both of these routes on a street map. For both of these cases, we placed

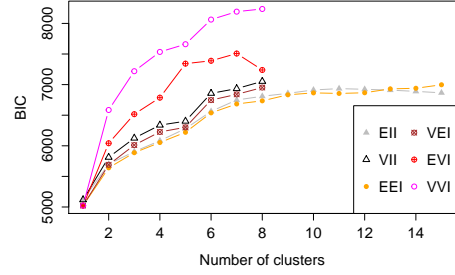


Figure 17. Estimated number of clusters

the phone in the central console between the driver and the front passenger seat. Over the course of next few days, we collected several test traces on the commute route from A to B by placing the phone at various positions inside the vehicle in an arbitrary orientation. We then compared these test traces with the initial data collected from both the actual and control route using dynamic time warping. Fig. (14) shows the normalized DTW distances calculated between the test cases and the original data from both the actual and control routes. A red circle represents normalized distance calculated between the test trace for a particular phone position in the vehicle and the initial data from the actual route. A blue triangle represents the distance between the same test trace and the initial data from the control route. It shows that under the normalized DTW distance, the test cases are consistently closer to the actual route as compared to the control route irrespective of the phone position and orientation in the vehicle. It is, therefore, possible to correctly recognize the actual route with the phone is located anywhere in the vehicle in an arbitrary orientation. We also used different devices to capture test traces from the actual route and compared these in a similar manner. Fig. (15) shows the normalized distances between the test traces from a Samsung S2, an iPhone 5s and an embedded sensor board (described later) and the initial data from the actual and control route captured with a Samsung S2. It shows that test traces are closer to the actual route under the normalized DTW distance. This demonstrates that our approach works across devices as well. We also evaluate cases where small detours are taken from the actual route while traveling from source A to destination B. Fig. (16) shows that even in the presence of these small detours, the traces taken from actual route are close to each other under DTW distance as compare to that from the control route. The DTW distance from the actual route gradually increases with the length of the detour.

### 4.2 Route Clustering

In order to demonstrate the feasibility of route mining, we collected a labelled dataset involving several typical journeys including commuting, leisure and other traveling undertaken over a course of two months in two different cities. We captured over 240Mbytes of sensor data from about 400 kilometers of driving under varying traffic conditions including urban, suburban and highway driving. This includes normal daily driving with lane changes, varying turning speeds

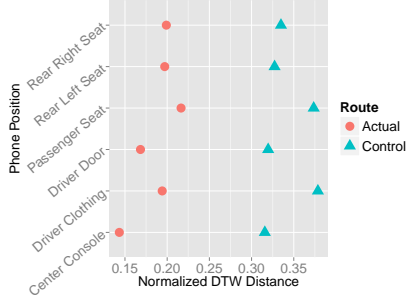


Figure 14. Phone position

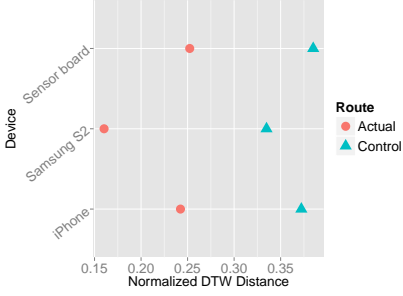


Figure 15. Different devices

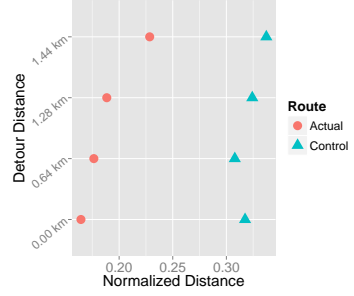


Figure 16. Small detours

and overtaking other vehicles. It also includes journeys that start or stop in car parks where turns vary between trips. For each journey, we collected accelerometer, gyroscope and GPS sensor data and recorded the source and destination of the journey. The dataset includes a total of 43 journeys over 15 routes (source destination pairs). The shortest route is about 4km where as the longest route is just over 90km long. Some parts of certain routes also overlap for different source or destination locations. For example, when routes originating at different source locations end in the same destination and when routes start at the same source but end at different destination locations.

We use k-medoids [21] to partition this dataset into clusters. It is a robust version of the popular k-means clustering algorithm and uses actual data points as cluster centers which is useful in our application because we intend to compare new routes with these clusters. However, any of the several clustering algorithms available in literature can be used for this partitioning. A system based threshold  $\tau$  can be defined where any cluster with more than  $\tau$  members is marked as a significant route. Thus the user will have to make at least  $\tau$  journeys on a route for it to be marked as a significant route of the user. Before we can cluster the routes with k-medoids, we have to determine  $k$ , the number of clusters in our dataset. One possible approach to this is to use finite mixture models [32] that view the data as originating from different processes each modeled as a Gaussian distribution. The number of clusters can then be estimated by varying the number of processes and model parameters and observing the Bayesian Information Criterion (BIC) [47] for model selection. Fig. (17) shows that BIC is maximized when the data is modeled as originating from 8 different processes among various model parametrizations. This suggests that there are 8 clusters in the dataset.

Fig. (18) shows the partitioning clusters generated by k-medoids with  $k = 8$  from the dissimilarity matrix  $D$  of normalized DTW distances between the time series of angular speeds from different journeys. It shows that the routes are organized in dense well separated clusters. Similar routes are clustered together into a single cluster or in nearby clusters. For example, cluster 1 contains all the journeys from source A to destination B and the journeys in reverse direction are grouped in the adjacent cluster 3. Three of the four routes grouped together in cluster 7 originate from one source whereas the fourth route has a different starting lo-

cation. However, all four routes have a common destination and therefore overlap with each other.

To evaluate this clustering more objectively, we compare it to GPS trajectories of these journeys. As the journeys can be of unequal length and partially overlapping, we have to create a mapping between each pair of trajectories using location data. DTW has been used for computing similarities between 2D trajectories [52] and GPS data [36] in trajectory analysis literature. When used with 2D (or 3D) location data, it creates a mapping between each location point  $p_1$  in trajectory  $T_1$  and location point  $p_2$  in the second trajectory  $T_2$  and then uses the supplied distance function (great circle distance for latitude longitude coordinates) between mapped points in  $T_1$  and  $T_2$  to calculate similarity between  $T_1$  and  $T_2$ . This gives us the intended output i.e. the larger the overlap between two trajectories, the smaller the dissimilarity and vice versa.

As we use geographic distances and the fact that the journeys took place in two different cities, initial clustering of GPS trajectories leads to two clusters, one in each city. We then perform a second round of clustering on routes in each of the two cities individually. We again use the same approach that we used for angular speed clustering. We use BIC to estimate the number of clusters in each city and then use k-medoids to perform clustering. This leads to 6 clusters in city A and 2 clusters in city B. These are shown in Fig. (19) and Fig. (20) respectively. While the shapes and separation between clusters of GPS trajectories are different from those based on angular speeds, the figures show that the overall partitioning is very similar. Clusters  $\{1, 2, \dots, 6\}$  of Fig. (19) correspond to the clusters with similar labels in Fig. (18) and clusters 1 and 2 of Fig. (20) correspond to clusters 7 and 8 of Fig. (18). A corrected rand index [42] and variation of information distance (VI) [34] are two measures that are extensively used to compare two different clustering outputs to see how similar or different they are. We compute both of these measures to compare the clusters based on angular speeds to those based on GPS trajectories. A rand index of  $r = 1$  and variation of information distance  $VI = 0$  between the two clusterings show that these clusterings are identical i.e. individual journeys or routes that are clustered together based on GPS trajectories are in the same clusters when partitioning is performed with gyroscope angular speeds. This shows that if we want to sense and later recognize routes or journeys, angular speeds from the gyroscope can provide us

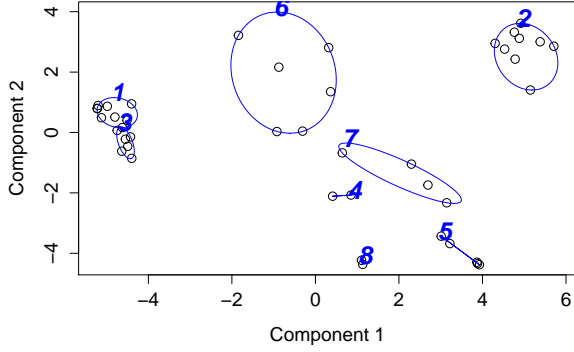


Figure 18. Clustering of routes based on angular speed

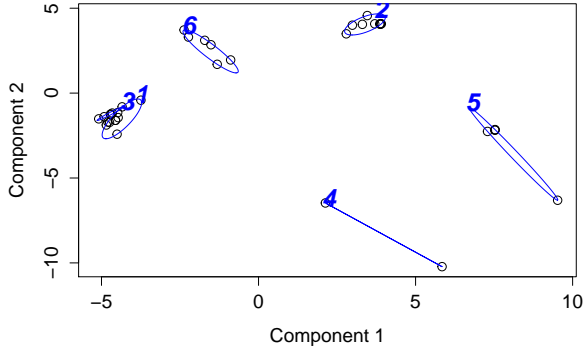


Figure 19. Clustering of routes based on GPS trajectories in city A

with the same information as the GPS sensor.

### 4.3 Grid Road Networks

In the last few subsections, we showed that it is possible to recognize a trip by inspecting variations of angular speeds experienced by the phone resulting from turns taken by the vehicle during the journey. But is it possible to use this approach in urban areas with Manhattan style grid based road network where turns at majority of the intersections are similar? We argue that on a grid type road network, a reasonably long route can be uniquely identified because it exhibits a unique *sequence of turns* even if individual turns are similar.

In order to show this, we use a dataset recently released by the New York metropolitan transportation authority containing information of real taxi trips in New York<sup>1</sup>. For each trip, the dataset contains travelled distance, pick up and drop locations and other metadata. According to 2009 National Household Travel Survey [46], the average length of a typical vehicle trip in a metropolitan area in US is 8.8 miles. We, therefore, take a random sample of 1000 taxi trips in New York from the dataset with an average distance of 9 miles. As the dataset does not contain actual routes taken by taxis,

<sup>1</sup>Taxi Trips [http://chriswhong.com/open-data/foil\\_nyc\\_taxi/](http://chriswhong.com/open-data/foil_nyc_taxi/)

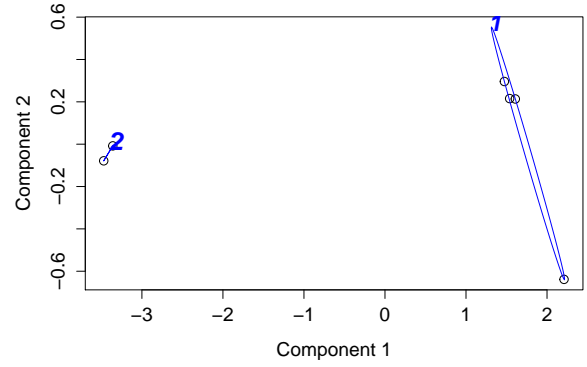


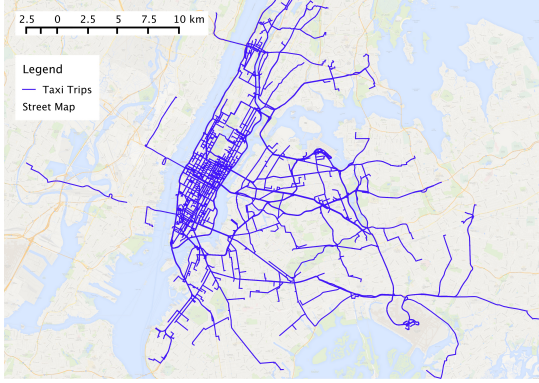
Figure 20. Clustering of routes based on GPS trajectories in city B

Table 1. Rules for encoding turning angles into alphabets

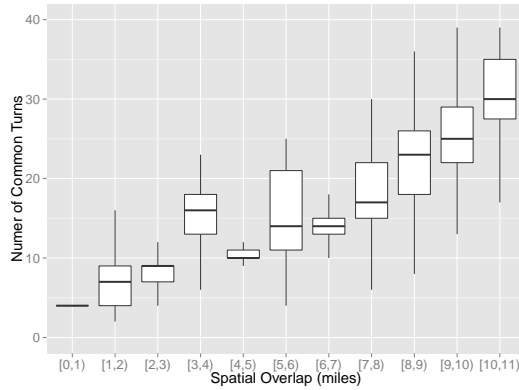
Encoding Rule	Turn Information	Encoded Alphabet
$-30^\circ \leq t < -15^\circ$	Slight Right	S
$t < -30^\circ$	Right	R
$15^\circ < t \leq 30^\circ$	Slight Left	T
$t > 30^\circ$	Left	L
$-15^\circ \leq t \leq 15^\circ$	Ignore	N/A

we use the A\* routing algorithm [14] on Open Street Map road network of New York to find road segments that form the shortest route between pickup and drop off locations for each trip. Given these shortest route road segments for each trip, we then compute turning angles for all turns along the computed route, encode these turns into alphabets according to the rules given in Table (1) and concatenate resulting characters into a string in the same order in which associated turns occur along the route. Thus for each trip, we have a trajectory in the form of road segments and a string with characters representing turns along this trajectory. We will refer to this as a maneuver string. Fig. (21) overlays all of these trajectories on a single layer showing that these trips span most of Manhattan and the surrounding urban area. In order to determine similarity between two trips based only on turns, we find the longest common sequence of characters between maneuver strings for each pair of trips. We then compare it to the spatial overlap between trajectories of same pair of trips. Fig. (22) plots the number of turns in the longest common sequence of turns against the spatial overlap for all possible combinations of 1000 taxi trips. It shows that for trajectories with no or little overlap (in the [0,1) miles interval), the number of turns in the longest common sequence of turns on average is just 4. Whereas the average number of turns in a typical trip trajectory in our sample is 22. This shows that trips of average length can be uniquely identified based solely on turn information even on grid type road networks.





**Figure 21. Taxi trips used for calculating route similarity in Manhattan New York**



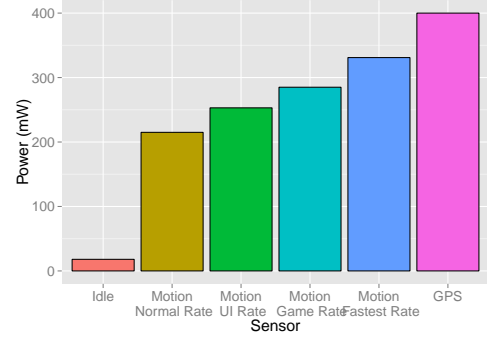
**Figure 22. Turn based route similarity in Manhattan**

## 5 Energy Evaluation

In the last few sections, we showed that a smartphone’s accelerometer and a gyroscope can be accurately used to sense and recognize routes frequently used by a driver. We now discuss first our implementation of the system on a smartphone and then our co-processor based implementation to then discuss their performance.

### 5.1 Smartphone Based System Evaluation

We start by investigating the energy consumption of different sensors on a smartphone. We use a Monsoon power monitor<sup>2</sup> to measure energy consumption of a Samsung S2 smartphone running Android OS while sampling the accelerometer and the gyroscope. The Android API defines four different sampling rates (Normal, UI, Game and Fastest) that an application developer can specify to sample from these sensors. We measure the energy consumption of the phone with each of these sampling rates. We also measure the energy consumption when the phone is in idle state and when it is sampling the GPS sensor. Fig. (23) compares the energy consumption of all of these states. It shows that even with the lowest sampling rate, the energy consumption of sampling the accelerometer and the gyroscope is slightly more than 50% of the energy consumed with the GPS sensor. Increasing the sampling rate to fastest leads to energy



**Figure 23. Energy consumption of motion sensors and GPS on Samsung S2 smartphone**

consumption that is more than 80% of that with the GPS sensor. These measurements were taken with the phone only sampling the sensors and no additional post processing or computations were performed on the sensor data. However, the sensor data from MEMS accelerometers and gyroscopes usually contains high frequency noise that requires filtering and post processing before it can be used in the application. This processing will lead to even higher energy consumption. This shows that using these sensors for sensing might not be an energy efficient option when compared to GPS.

Several software based duty cycling approaches have been proposed [31, 38] to address high energy consumption of sensing on smartphones, however, these still do not lead to significant energy savings due to high energy consumption of the main processor of a smartphone [39]. A more effective approach is to use a sensing system with its own dedicated low power processor. This approach has been recently adopted by device manufacturers and co-processors have been integrated in modern smartphones like Apple’s iPhone 5s<sup>3</sup> and Motorola’s Moto X<sup>4</sup>. Apple’s iPhone 5s continuously tracks users activities including walking, running and detecting that the user is in a motorized transport using motion sensors. Motorola’s Moto X performs continuous voice sensing and recognition. However, these systems are closed and do not offer developers the ability to program or leverage these co-processors for their own applications. In order to demonstrate the feasibility and energy efficient of our approach on such dedicated co-processors, we use a separate sensing board that includes the necessary sensors and a low power processor and can be connected to a smartphone either through bluetooth or USB interface. However, our approach can be integrated into devices with internal co-processors if the manufacturers open up these systems. This could lead to say an iPhone not only detecting that a user is in a motorized transport but also recognizing that the user is traveling on his or her usual commute route using only the motion sensors. This system and its performance are described next.

<sup>3</sup><https://www.apple.com/iphone-5s/specs>

<sup>4</sup><http://www.motorola.com/us/FLEXR1-1/moto-x-specifications.html>

<sup>2</sup><http://www.monsoon.com/LabEquipment/PowerMonitor/>

	Predicted	
Actual	Driving	Null
Driving	49.7%	0.7%
Null	1.5%	48.1%

**Table 2. Confusion matrix for driving detection**

## 5.2 Co-processor Based System Evaluation

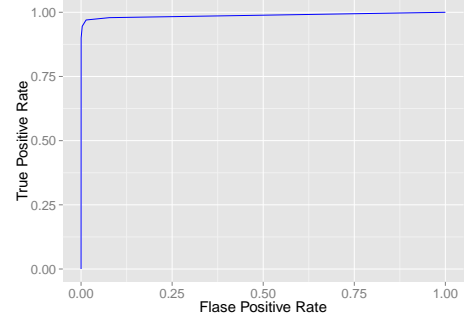
As we mentioned, the coprocessors on recently released phones are not programmable so to test the performance of our system with a coprocessor we resorted to use different programmable hardware shown in Fig. (24).

Our embedded sensor board is built around a 16bit MSP430F5528 processor with 8kB SRAM and 128kB internal program flash. It provides a rich collection of interfaces including I2C and SPI buses, UARTs and a built in USB 2.0 interface with its separate 2kB SRAM. The processor also includes a 16bit hardware multiplier. A bluetooth 2.0 module is connected to one of the UARTs of MSP430. The processor is also connected via I2C to a sensor chip MPU-9150 that includes motion sensors. It also has an internal buffer to temporarily store sensor data and generates interrupts to MSP430 to indicate when the data is ready allowing the processor to enter into low power sleep modes between samples. Each of the motion sensors can be individually switched on and off and sampled at rates varying from 10Hz to 100Hz. Fig. (25) shows a block diagram of our embedded sensor board. This sensor board is connected to a Samsung S2 phone through an OTG USB cable allowing an Android application to communicate with it through the Android USB framework. Fig. (32) shows an overview of the entire system with different software and hardware components. Here we describe these components in detail.

**Sensor** We use MPU-9150 as our main sensor. It integrates a high quality accelerometer, a gyroscope and a magnetometer and outputs 16bit data for each of these sensors at configurable sampling rates. As we mentioned earlier, MEMS sensor data usually contains high frequency noise and requires filtering before it can be used. The chip has an internal digital low pass filter with a configurable cut-off frequency that frees up the processor from digital filter computations. We set up this filter with a cut-off frequency of 42Hz. The chip also includes a pedometer to detect if a user is walking and step counting. This can be used to detect if a user has finished his or her journey and has started walking. We, however, use a simple activity recognition algorithm running on MSP430 to detect these transitions.

**Low Power Processor** MSP430 runs a simple activity recognition algorithm based on a decision tree to detect when a user starts and stops driving. When it detects that the user is driving, it samples, aggregates and buffers sensor data from MPU-9150, estimates the gravity vector and pushes aggregated data to the phone if realtime route recognition is required. Otherwise the data is stored in the serial flash and can be used later for route mining.

There is a significant amount of work on activity recognition on smartphones and wearable devices [7, 28, 25, 48]. Recent devices like Apple’s iPhone 5s already include this capability. Our aim over here is just to demonstrate that, by



**Figure 27. ROC curve for driving activity recognition**

using one of these techniques, it is possible to perform continuous sensing and to trigger route sensing automatically without user interaction in an energy efficient manner. In order to train a decision tree for driving detection, we collected accelerometer data from the phones of two users who do not use any motorized transport. We instrumented the phones to sample the accelerometer for 8s after every two minutes and save the data on the phone. This data was collected continuously over 5 days to capture the entire range of activities that a user might perform apart from driving. This formed the null class for our model training. We used the accelerometer data from the dataset discussed in Section(3.4) that was collected during driving as the primary class. Using these two labelled classes, we trained a simple decision tree with per axis mean and variance of acceleration as features. We split the dataset in a 60% training and 40% testing data making sure that both classes are represented equally in both data partitions. We trained the classifier on the training data and then validated its performance on the testing data. Table (5.2) shows the confusion matrix when we run the trained classifier on the testing data never seen by the classifier during training. It shows that both classes are equally represented in the testing data and that the classification error is very small. Fig. (27) shows a ROC curve for this classifier.

MSP430 samples the accelerometer (the gyroscope is disabled) at 10Hz, computes features over a 2s window and runs the trained decision tree driving detection. When driving is detected, it configures the MPU-9150 to sample from both the accelerometer and the gyroscope at 50Hz. It computes a running average over 1s on the gyroscope data and after one second transfers the mean  $x$ ,  $y$  and  $z$  angular speed components to a separate buffer. We use a 1440 bytes buffer to store up to 4 minutes of angular speed data. A running mean of the accelerometer data is also calculated over these 4 minutes to estimate the gravity vector. MSP430 then enables its USB interface and sends buffered angular speed data and gravity vector to the phone. The USB interface is then disabled to save energy.

**Smartphone Processor** On the smartphone, we developed an application that registers itself with the Android OS with MSP430’s USB vendor identifier (VID) and product identifier (PID). When the MSP430 enables its USB interface, a Broadcast Intent triggers Android OS to start our application on the phone even if the phone is in idle standby

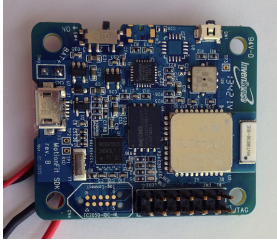


Figure 24. Sensor board

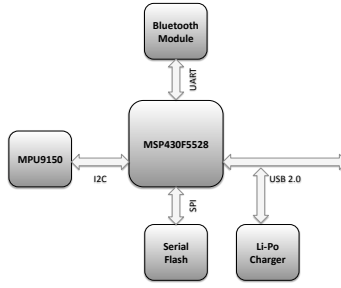


Figure 25. Board components

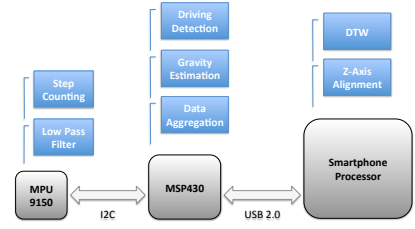


Figure 26. System overview

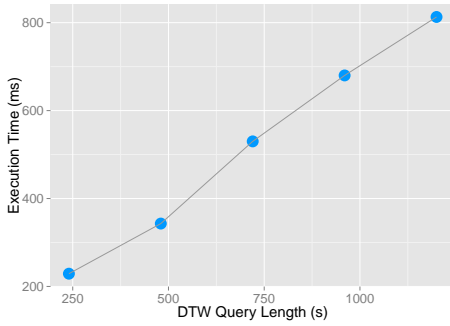


Figure 28. DTW execution time on Samsung S2

state. We implemented the Z-Axis alignment discussed in Section (3.1) and Dynamic Time Warping discussed in Section (3.2) in this application. It transfers the data from the sensor board to the phone and then either saves it to storage or runs these algorithms to perform realtime route recognition on the phone. When configured to perform realtime route recognition, it executes Z-Axis alignment and then computes normalized DTW distance between current angular speed trace and cluster centers in its database. As new angular speed data is received from the sensor board after every 4 minutes, it is combined with the previously received data and the DTW distance is recalculated. This is only done up to 5 times (20 minutes of angular speed data) and if no match is found with any of the saved cluster centers, it is classified as new route not seen before and marked for storage. Fig. (28) shows the execution time for calculating DTW distance of current angular speed data as it is received from the sensor board with a previous cluster center representing a 24 minute journey. It shows that execution time increases linearly as new data arrives in from the sensor board and the DTW query length increases. It takes slightly more than 800ms to compare maximum query length of 20 minutes with a cluster center. Z-axis alignment can be executed on each batch of data individually without combining it with the previous batch. On Samsung S2, our quaternion based z-axis alignment executes on average in 4ms for 4 minutes worth of angular speed data.

**Energy Consumption** We now investigate the energy consumption of our route sensing and recognition system.

We use the Monsoon power monitor to measure current and power consumption of the system. With the driving detection classifier running on the embedded sensor board, MPU-9150 generates a 10Hz interrupt triggering the MSP430 to exit the LPM0 low power mode to retrieve the accelerometer data from the chip and to execute feature extraction and classification. In this sensing state, the sensor consumes 5mW of power. This is less than one third of the energy consumption of Samsung S2 phone in its idle state and two orders of magnitude smaller than sampling motion sensors or GPS on the phone. When the sensor board detects that the user is driving, it enables the gyroscope and configures a sampling rate of 50Hz for the both the accelerometer and the gyroscope. With data aggregation and buffering, gravity estimation and the activity classifier, the sensor board consumes 25mW of power. However, now after every 4 minutes, it enables its USB interface and transfers buffered data to the phone. Fig. (29) shows a power trace captured with the power monitor for one of these USB transfers. It shows that power consumption increases to 59mW when the USB interface is enabled and drops back to 25mW when the transaction is complete and the interface is disabled to save energy. It lasts for about 4s mainly because the sensor board has to wait for the phone to load the Android application to receive data. Although, our sensor board has a bluetooth module, we decided not to use for data transfer it due to its high energy consumption.

When 4 minutes of buffered angular speed data and gravity vector is transferred to the phone, z-axis alignment is executed on the phone and takes on average 4ms on Samsung S2. In order to capture the power consumption of this computation on the power monitor, we repeat it 500 times in a tight loop on the phone. Fig. (30) shows the power consumption of Samsung S2 during this loop repeating z-axis alignment computation for 500 times. It shows that the mean power consumption of z-axis alignment is about 900mW. Fig. (31) shows the power consumption of Samsung S2 when executing DTW with two 20 minute long routes i.e. 1200 angular speed data points (1 per second) for each route. This is the maximum length DTW query that our system ever executes. Any routes not matched within 20 minutes of driving can safely be assumed to be new routes not seen before and therefore do not require DTW based comparison. Fig.

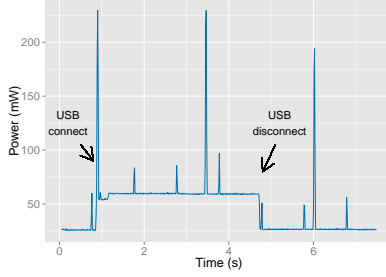


Figure 29. USB transaction

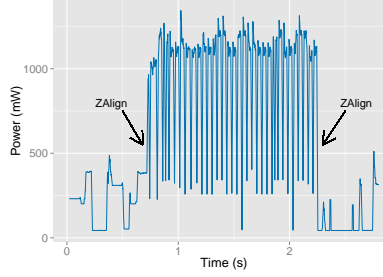


Figure 30. Z-Alignment

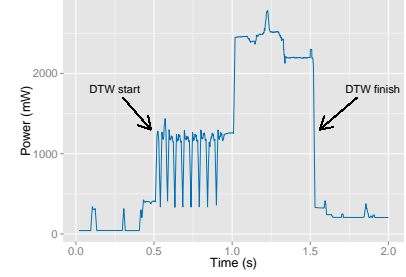


Figure 31. DTW computation

Function	Power (mW)	Execution Time (s)	Execution Interval (s)	Mean Power (mW)
Sensing	25	-	-	25
USB Transfer	59	4	240	0.97
Z Alignment	910	0.004	244	0.02
DTW	1560	2 - 6.6	-	28.01
Total				54

Table 3. Mean power consumption of system components

(31) shows that the mean power consumption of Samsung S2 for this task is about 1500mW.

Table (5.2) lists the power consumptions of different system components along with their execution times and intervals if any apply to them. The row marked as Sensing refers to the power consumption of the sensor board when it is in route sensing state. This lasts throughout the route and therefore has no execution time or interval associated with it. USB transfers occur every 4mins (240s) and last only for 4s. Thus the mean power consumption of this task over the entire duration of the route is 0.97mW. Similarly the mean power consumption of z-axis alignment over the entire duration of route is 0.02mW. DTW executes only up to a maximum of 5 times during the entire route. The total execution time however depends on the amount of buffered data received so far and the number of clusters to compare with. Based on our dataset, we assume a total of eight clusters which gives us a running time of 2 to 6.6s as the query length grows from minimum to maximum. The mean power consumption of DTW execution over the entire duration of a 20min route is thus about 28mW. For longer routes, the mean power consumption will be lower than this as the cost of running DTW is averaged over the entire duration of the route. The total average power consumption of the complete route sensing and recognition (sum of all components) is about 54mW.

Fig. (32) compares the power consumption of our system with the energy consumption of Samsung S2 in its idle state, sampling motion sensors on the phone and the GPS sensor. It shows that the energy consumption of our system (5mW) in its continuous activity recognition state is two orders of magnitude less than sampling either the motion sensors on the phone or the GPS. When our system is in its route sensing and recognition state, the mean power consumption (54mW) is still an order of magnitude less than sampling either of the sensors on the phone. To put this in context, the mean power

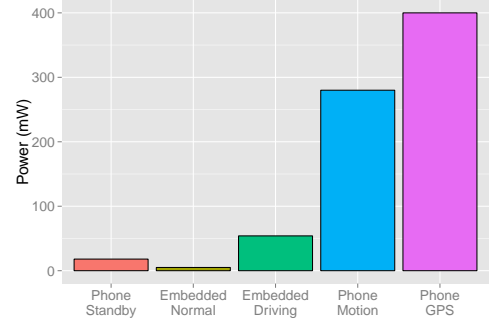
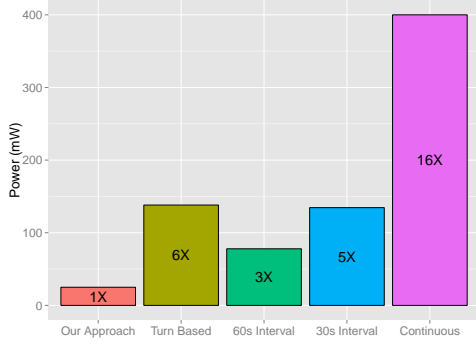


Figure 32. Power consumption of our approach compared to phone's motion and GPS sensors

consumption of sending a single text message and replying to an email over WiFi is about 300mW and 432mW [4] respectively. For a one hour journey, our approach leads to less than 0.01% usage of a typical 1650mAh phone battery. For phone motion sensors or GPS, we only included the energy consumption of sampling the sensor from the main processor without any further computation or calculations. Adding the cost of any post processing or computation will increase the power consumption of any system based on these sensors even further.

Now we compare the energy consumption of our gyroscope based route sensing (without DTW and route recognition) with that of sampling a duty cycled GPS sensor. Duty cycling a typical GPS receiver is usually not straight forward due to long and unpredictable time required to acquire a fix and loss of hot start state soon after it is switched off [41]. Reducing the sampling interval of GPS beyond 30s in a moving vehicle also makes it difficult to infer the actual route taken by the vehicle with map matching [37]. We duty cycle the GPS on Samsung S2 with a 30s interval while traveling on the path shown in Fig. (13) over a week and measure GPS fix times. The mean acquisition time of GPS in this duty cycled setting is 8s. We exclude the time required for initial fix and any measurements where the GPS lost the fix after initially acquiring it (which is not uncommon in dense urban areas) thus making the comparison more favorable for the GPS. Power consumption of GPS in this duty cycled setting measured using Monsoon power monitor is 425mW. Assuming an initial GPS fix time of 60s, Fig. (33) compares the energy consumption of our route sensing with both contin-





**Figure 33. Power consumption of our approach as compared to duty cycled GPS**

uous and duty cycled GPS sensor. It shows that even under good conditions (small initial fix time and no loss of signal), energy consumption of the GPS sensor duty cycled with 30s and 60s intervals is five and three times of that of gyroscope based sensing. It also shows that a turn based duty cycling approach that detects turns using gyroscope and then triggers the GPS (mean time between turns in our dataset is 27s) is six times the gyroscope based route sensing. Another approach for reducing GPS energy consumption is to offload GPS signal decoding and location computation to a cloud service [29], however, these techniques are only suitable for delay tolerant applications and cannot be used (without incurring the data transfer overhead) for real-time route recognition while the user is in transit to trigger various actions e. g. sending notifications to friends/family, switching heating systems etc.

## 6 Discussion

We presented a system that can infer a user’s significant journeys by using only the accelerometer and gyroscope of user’s phone. Once the system has identified significant routes, it can either infer or gather more information about these journeys. For example, for a typical user, it can infer that the most frequently repeated last route of the day is a journey to home. It can also present a time annotated list of the most frequently taken routes to the user and request a label for these journeys. This approach is quite typical and frequently used by significant place learning systems. For example, Google Now [2], a context and place learning application by Google, requests users to mark home and work locations after these have been identified by the system. This label based approach is sufficient for some of the applications that we think will benefit from route learning. For example, a smart home heating application [3] can present a list of route labels provided by this system to the user and ask to choose journeys that should start or stop the heating. When any of the selected journeys are detected by the system, it can inform the application with a callback and the application can take necessary actions. Similarly, another application that we envisage is route based alerts or messaging. The user can select a previously labeled route to set up a text message to be sent to family or friends when that journey is detected to inform them that the user is on his way or

to send a message when a routine journey is taking unusually long say due to a traffic jam.

There is a second class of applications, however, that require the actual physical route. For example, a traffic alert application that informs the user that there is a disruption on his regular route (an accident or road works etc) requires the actual physical path to provide this type of information to the user. For these applications, our system can annotate the significant routes with location data say from GPS. This approach is still energy efficient as compared to using only location sensing for route learning. It acquires location data that has high energy cost *once only* for those routes that have already been identified as significant as opposed to acquiring location data for all the user journeys only to keep some of it and discarding the rest. This is again something that is used by place learning systems for automatically annotating semantic places with location data.

Another more sophisticated approach that the system can use is to estimate the turning angles by integrating angular speed and measure times between turns for a previously identified significant journey. Given these turning angles, times between turns and the source destination pair (as significant places are already sensed by place learning features on smartphones) of the journey, determine which one of all the possible routes between these two locations was taken by comparing the turns and times between turns from a GIS mapping service like Open Street Map by using a HMM based map matching approach [12].

## 7 Related Work

Dynamic Time Warping (DTW) has been used before in a number of settings, even in the sensing domain. Johnson et. al. [19] and Eren [9] describe a DTW based classifier that can be trained with various driving maneuvers by using the accelerometer and the gyroscope data. They use it to detect the type of the driving maneuver and whether it is being executed in a normal or an aggressive manner to sense the driving style of the user. Unlike this approach we use DTW to match entire routes and not to detect user driving behaviors on a limited set of maneuvers.

Chandrasekaran et. al. [5] also use Dynamic Time Warping (DTW) on the radio signal strength obtained from the GSM radio interface of the smartphone traveling in a vehicle. They compare it with a signal profile captured from the same road segment at a known speed to estimate short term vehicle speed variations. Liu et. al. [30] build a library of gesture templates from an accelerometer in a smartphone or a consumer device like Wii Remote and then use DTW for recognizing these gestures.

Mobile sensing has also been used in several other applications related to vehicles and traveling. For example, a number of studies have used accelerometers in smartphones to monitor road surface conditions [35, 10]. Others have used phone sensors to monitor driver behavior [8, 19, 9]. White et. al. [54] propose to use the accelerometers in phones for automatic traffic accident detection and requesting emergency help for vehicle occupants. Wang et. al. [53] use the accelerometer and the gyroscope of the phone during driving to detect whether the phone is on the driver side or passen-



ger side of the vehicle. All of these studies indicate that it is possible to use smartphone sensors to sense and monitor the subtle forces experienced by the phones while driving in a vehicle.

Both Android and iOS operating systems have embedded place learning features. Google Now [2], the context sensing core of Android OS, learns a user's home and work locations and then occasionally shows travel times between these based on the calculated shortest route. However, there are usually more than one routes available between different locations and which one of these is relevant to the user can only be determined by performing sensing which these applications do not appear to do probably to save energy. Some users have a personal preference or local knowledge and therefore prefer a particular route as opposed to the shortest route. Some users visit other locations, say to pick up a family member, while traveling home.

There is also a large body of research on inertial navigation that uses inertial sensors like accelerometer, gyroscope and magnetometer to estimate and track the position of the moving device by performing sensor fusion in a Kalman or a Particle filter. However, the low cost MEMS inertial sensors used in consumer devices like smartphones exhibit noise characteristics that make them unsuitable for an inertial only navigation. Woodman [55] and Tan et. al. [49] show that the position error of an MEMS based inertial navigation system grows to more than 100m in less than a minute. MEMS based inertial navigation systems, therefore, require periodic external reference information [49, 12] to keep this error bounded.

Thiagarajan [50] use the GSM radio interface of a smartphone along with the embedded accelerometer and magnetic sensor to estimate phone's trajectory in an energy efficient manner as compared to GPS. This approach, however, requires GSM wardriving data to pre-train the system. Also the software interface to GSM radios is usually controlled by proprietary drivers that do not expose the complete GSM fingerprint (neighboring cell towers) on most devices [26]. Krumm [23] and Froehlich [11] also present destination and route prediction from previously collected trips from GPS receivers in vehicles. Our approach, however, relies on low power sensors in user's smartphone.

## 8 Conclusion

We have proposed an approach which uses solely the accelerometer and gyroscope of the user's phone to detect repeated driving routes. With the advance of modern phones with embedded co-processor technology, our approach is one order of magnitude more efficient than any GPS based solution. Our future work involves generalizing the approach of significant journeys beyond driving to other modes of transport, for example cycles and motor bikes, that do not have a car like constrained movement model and exploring the possibility of implementing all the software components on embedded co-processor.

## 9 Acknowledgments

This research has been funded by the EPSRC Innovation and Knowledge Centre for Smart Infrastructure and Construction project (EP/K000314).

## 10 References

- [1] Android developers guide. [http://developer.android.com/guide/topics/sensors/sensors\\_motion.html](http://developer.android.com/guide/topics/sensors/sensors_motion.html).
- [2] Google Now. <http://www.google.com/landing/now>.
- [3] Tado smart heating. <http://www.tado.com>.
- [4] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.
- [5] G. Chandrasekaran, T. Vu, A. Varshavsky, M. Gruteser, R. Martin, J. Yang, and Y. Chen. Tracking vehicular speed variations by warping mobile phone signal strengths. In *Pervasive Computing and Communications (PerCom), 2011 IEEE International Conference on*, pages 213–221, March 2011.
- [6] Y. Chon, N. D. Lane, F. Li, H. Cha, and F. Zhao. Automatically characterizing places with opportunistic crowdsensing using smartphones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 481–490, New York, NY, USA, 2012. ACM.
- [7] T. Choudhury, S. Consolvo, B. Harrison, J. Hightower, A. LaMarca, L. Legrand, A. Rahimi, A. Rea, G. Bordello, B. Hemingway, P. Klasnja, K. Koscher, J. Landay, J. Lester, D. Wyatt, and D. Haehnel. The mobile sensing platform: An embedded activity recognition system. *Pervasive Computing, IEEE*, 7(2):32–41, April 2008.
- [8] J. Dai, J. Teng, X. Bai, Z. Shen, and D. Xuan. Mobile phone based drunk driving detection. In *Pervasive Computing Technologies for Healthcare (PervasiveHealth), 2010 4th International Conference on-NO PERMISSIONS*, pages 1–8, March 2010.
- [9] H. Eren, S. Makinist, E. Akin, and A. Yilmaz. Estimating driving behavior by a smartphone. In *Intelligent Vehicles Symposium (IV), 2012 IEEE*, pages 234–239, June 2012.
- [10] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan. The pothole patrol: Using a mobile sensor network for road surface monitoring. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, MobiSys '08, pages 29–39, New York, NY, USA, 2008. ACM.
- [11] J. Froehlich and J. Krumm. Route prediction from trip observations. In *Society of Automotive Engineers (SAE) 2008 World Congress*, 2008.
- [12] S. Guha, K. Plarre, D. Lissner, S. Mitra, B. Krishna, P. Dutta, and S. Kumar. Autowitness: Locating and tracking stolen property while tolerating gps and radio outages. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 29–42, New York, NY, USA, 2010. ACM.
- [13] J. Han, E. Owusu, L. Nguyen, A. Perrig, and J. Zhang. Accomplice: Location inference using accelerometers on smartphones. In *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on*, pages 1–9, Jan 2012.
- [14] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, July 1968.
- [15] S. Hemminki, P. Nurmi, and S. Tarkoma. Accelerometer-based transportation mode detection on smartphones. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, SenSys '13, pages 13:1–13:14, New York, NY, USA, 2013. ACM.
- [16] F. Hermans and E. Tsiporkova. Merging microarray cell synchronization experiments through curve alignment. *Bioinformatics*, 23(2):e64–e70, 2007.
- [17] J. Hightower, S. Consolvo, A. LaMarca, I. Smith, and J. Hughes. Learning and recognizing the places we go. In *Proceedings of the 7th International Conference on Ubiquitous Computing*, UbiComp'05, pages 159–176, Berlin, Heidelberg, 2005. Springer-Verlag.
- [18] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, Sept. 1999.
- [19] D. Johnson and M. Trivedi. Driving style recognition using a smartphone as a sensor platform. In *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*, pages 1609–1615, Oct 2011.
- [20] J. F. Kaiser and W. A. Reed. Data smoothing using low pass digital filters. *Review of Scientific Instruments*, 48(11):1447–1457, 1977.
- [21] L. Kaufman and P. Rousseeuw. Clustering by means of medoids. *Statistical Data Analysis Based on the L1-Norm and Related Methods*,

- pages North-Holland, 1987.
- [22] D. H. Kim, Y. Kim, D. Estrin, and M. B. Srivastava. Sensloc: Sensing everyday places and paths using less energy. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 43–56, New York, NY, USA, 2010. ACM.
  - [23] J. Krumm and E. Horvitz. Predestination: Inferring destinations from partial trajectories. In *Proceedings of the 8th International Conference on Ubiquitous Computing*, UbiComp'06, pages 243–260, Berlin, Heidelberg, 2006. Springer-Verlag.
  - [24] J. B. Kuipers. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace and Virtual Reality*. Princeton University Press, Aug. 2002.
  - [25] J. R. Kwapisz, G. M. Weiss, and S. A. Moore. Activity recognition using cell phone accelerometers. *SIGKDD Explor. Newsl.*, 12(2):74–82, Mar. 2011.
  - [26] A. LaMarca, Y. Chawathe, S. Consolvo, J. Hightower, I. Smith, J. Scott, T. Sohn, J. Howard, J. Hughes, F. Potter, J. Tabert, P. Powledge, G. Borriello, and B. Schilit. Place lab: Device positioning using radio beacons in the wild. In *Proceedings of the Third International Conference on Pervasive Computing*, PERVASIVE'05, pages 116–133, Berlin, Heidelberg, 2005. Springer-Verlag.
  - [27] L. J. Latecki, V. Megalooikonomou, Q. Wang, and D. Yu. An elastic partial shape matching technique. *Pattern Recognition*, 40(11):3069–3080, 2007.
  - [28] J. Lester, T. Choudhury, and G. Borriello. A practical approach to recognizing physical activities. In *Proceedings of the 4th International Conference on Pervasive Computing*, PERVASIVE'06, pages 1–16, Berlin, Heidelberg, 2006. Springer-Verlag.
  - [29] J. Liu, B. Priyantha, T. Hart, H. S. Ramos, A. A. F. Loureiro, and Q. Wang. Energy efficient gps sensing with cloud offloading. In *Proceedings of the 10th ACM Conference on Embedded Networked Sensor Systems*, SenSys '12, pages 85–98, New York, NY, USA, 2012. ACM.
  - [30] J. Liu, L. Zhong, J. Wickramasuriya, and V. Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing*, 5(6):657–675, 2009. PerCom 2009.
  - [31] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell. The jigsaw continuous sensing engine for mobile phone applications. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 71–84, New York, NY, USA, 2010. ACM.
  - [32] G. J. McLachlan and D. Peel. *Finite mixture models*. Wiley Series in Probability and Statistics, New York, 2000.
  - [33] A. Mednis, G. Strazdins, R. Zviedris, G. Kanonirs, and L. Selavo. Real time pothole detection using android smartphones with accelerometers. *2013 IEEE International Conference on Distributed Computing in Sensor Systems*, 0:1–6, 2011.
  - [34] M. Meil. Comparing clusterings: an information based distance. *Journal of Multivariate Analysis*, 98(5):873–895, 2007.
  - [35] P. Mohan, V. N. Padmanabhan, and R. Ramjee. Nericell: Rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems*, SenSys '08, pages 323–336, New York, NY, USA, 2008. ACM.
  - [36] B. Morris and M. Trivedi. Learning trajectory patterns by clustering: Experimental studies and comparative evaluation. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 312–319, June 2009.
  - [37] P. Newson and J. Krumm. Hidden markov map matching through noise and sparseness. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '09, pages 336–343, New York, NY, USA, 2009. ACM.
  - [38] J. Paek, J. Kim, and R. Govindan. Energy-efficient rate-adaptive gps-based positioning for smartphones. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 299–314, New York, NY, USA, 2010. ACM.
  - [39] B. Priyantha, D. Lymberopoulos, and J. Liu. Littlerock: Enabling energy-efficient continuous sensing on mobile phones. *IEEE Pervasive Computing*, 10(2):12–15, 2011.
  - [40] L. Rabiner and B.-H. Juang. *Fundamentals of Speech Recognition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.
  - [41] H. S. Ramos, T. Zhang, J. Liu, B. Priyantha, and A. Kansal. Leap: A low energy assisted gps for trajectory-based services. In *13th ACM International Conference on Ubiquitous Computing (UbiComp)*. ACM, September 2011.
  - [42] W. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.
  - [43] H. Sakoe and S. Chiba. A dynamic programming approach to continuous speech recognition. In *Proceedings of the Seventh International Congress on Acoustics, Budapest*, volume 3, pages 65–69, Budapest, 1971. Akadémiai Kiadó.
  - [44] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, (1):43–49, 1978.
  - [45] Y. Sakurai, C. Faloutsos, and M. Yamamuro. Stream monitoring under the time warping distance. *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, 0:1046–1055, 2007.
  - [46] A. Santos, N. McGuckin, H. Y. Nakamoto, D. Gray, and S. Liss. Summary of Travel Trends: 2009 National Household Travel Survey. Technical Report FHWA-PL-11-022, U.S. Department of Transportation, June 2011.
  - [47] G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 03 1978.
  - [48] L. Stenneth, O. Wolfson, P. S. Yu, and B. Xu. Transportation mode detection using mobile phones and gis information. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '11, pages 54–63, New York, NY, USA, 2011. ACM.
  - [49] G. Tan, M. Lu, F. Jiang, K. Chen, X. Huang, and J. Wu. Bumping: A bump-aided inertial navigation method for indoor vehicles using smartphones. *IEEE Transactions on Parallel and Distributed Systems*, 99(Preliminary):1, 2013.
  - [50] A. Thiagarajan, L. Ravindranath, H. Balakrishnan, S. Madden, and L. Girod. Accurate, low-energy trajectory mapping for mobile devices. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI '11, pages 20–20, Berkeley, CA, USA, 2011. USENIX Association.
  - [51] P. Tormene, T. Giorgino, S. Quaglini, and M. Stefanelli. Matching incomplete time series with dynamic time warping: An algorithm and an application to post-stroke rehabilitation. *Artif. Intell. Med.*, 45(1):11–34, Jan. 2009.
  - [52] M. Vlachos, D. Gunopulos, and G. Das. Rotation invariant distance measures for trajectories. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 707–712, New York, NY, USA, 2004. ACM.
  - [53] Y. Wang, J. Yang, H. Liu, Y. Chen, M. Gruteser, and R. P. Martin. Sensing vehicle dynamics for determining driver phone use. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 41–54, New York, NY, USA, 2013. ACM.
  - [54] J. White, C. Thompson, H. Turner, B. Dougherty, and D. Schmidt. Wreckwatch: Automatic traffic accident detection and notification with smartphones. *Mobile Networks and Applications*, 16(3):285–303, 2011.
  - [55] O. J. Woodman. An introduction to inertial navigation. Technical Report UCAM-CL-TR-696, University of Cambridge, Aug. 2007. Technical Report.